The Impact of the Web Prefetching Architecture on the Limits of Reducing User's Perceived Latency

Josep Domènech, Julio Sahuquillo, José A. Gil, Ana Pont
Department of Computer Engineering (DISCA)
Universitat Politècnica de València (Spain)
jodode@doctor.upv.es; {jsahuqui, jagil, apont}@disca.upv.es

***Abstract*— Web prefetching is a technique that has been researched for years to reduce the latency perceived by users. For this purpose, several web prefetching architectures have been used, but no comparative study has been performed to identify the best architecture dealing with prefetching. This paper analyzes the impact of the web prefetching architecture focusing on the limits of reducing the user's perceived latency. To this end, the factors that constrain the predictive power of each architecture are analyzed and these theoretical limits are quantified. Experimental results show that the best element of the web architecture to locate a single prediction engine is the proxy, whose implementation could reduce the perceived latency up to 67%. Schemes for collaborative predictors located at diverse elements of the web architecture are also analyzed. These predictors could dramatically reduce the perceived latency, reaching a potential limit of about 97% for a mixed proxy-server collaborative prediction engine.**

I. Introduction

In recent years, a high amount of research efforts has been focused on reducing the latency perceived by users when browsing the Web. One of the techniques researched is web prefetching, which preprocesses user's requests before the user actually demands them. Web prefetching related studies are performed through different prefetching architectures. For instance, [1] proposes the proxy servers to push web objects to low bandwidth clients. A proxy where user's requests are predicted and requested to the server in advance is described in [2], whereas [3] proposes different levels of proxies and servers collaborating to predict users' requests. Other works [4], [5] implement a system where servers predict future accesses and then, clients download those objects in advance.

These works provide results in different conditions and architectures, but do not show insights about how far these results are from the maximum benefits achievable by prefetching techniques. These benefits depend on several factors as well as the considered web architecture. An initial tentative to quantify the limits for latency reduction of web caching and prefetching is presented in [6]. Unfortunately, they combine their benefits and they only focus on the proxy. In addition, some of their assumptions do not reamain valid in current web.

This paper analyzes the impact of the web prefetching architecture on the limits of reducing the user's perceived latency. To do so, we examine the factors that limit the predictive power of each prefeching architecture and quantify these theoretical limits both in amount of requests that can be predicted and in user's perceived latency that can be reduced. This study addresses two main issues: i) to identify the best architecture to reduce user's perceived latency when performing prefetch, and ii) to provide insights about the goodness of a given proposal by comparing it to the performance limits.

[1]This work has been partially supported by Spanish Ministry of Education and Science and the European Investment Fund for Regional Development (FEDER) under grant TSI 2005-07876-C03-01.

II. Generic Web Prefetching Architecture

Proposed prefetching systems are usually based on a generic web architecture, which is extended to implement both prediction and prefetching engines in the same or different elements of the system. In this section, we describe the elements that compose a generic web architecture and how it can be extended and used to carry out prefetching.

A. Generic Web Architecture

There are two main elements in a generic web architecture: i) user agents, or clients, that is, the software employed by users to access the Web, and ii) web servers, which contain the information that users demand. The generic web architecture works as follows. The human user tells the client which page he wants to retrieve by writing down a URI or by clicking a hyperlink of a previously loaded page. Then, the client demands the page to the server by requesting each object that the page consists of, and finally, the whole page is displayed to the user. Optionally, there may be more elements between the clients and the servers.A proxy is usually located near a group of clients to cache the most popular objects accessed by that group. By doing so, they reduce the user's perceived latency and the network traffic between the proxy and the servers. Surrogates, also called reverse proxies, are proxies located by the server side. They cache the most popular server responses and are usually transparent for the clients, which access to the surrogate as if they access to the web servers.

B. Prediction Engine

The prediction engine is the part of the prefetching system aimed at guessing the following user's accesses. This engine can be located at any part of the web architecture: clients [7], proxies [2], [1], and servers [8], [5], [4], or even in a collaborative way between several elements [3]. To make the predictions, a high amount of algorithms that learn from the past access patterns to predict future ones have appeared in the literature, e.g., [4], [3]. These patterns of user's accesses

differ depending on the element of the architecture in which the prediction engine is implemented. For instance, a predictor located at the web server can only gather transitions between pages of the same server, but not cross-server transitions.

The output of the prediction engine is the hint list, which is composed of a set of URIs which are likely to be requested by the user in a near future. Nevertheless, this list can also be composed of a set of servers if only the connection is going to be prefetched.

Notice that it is equivalent to predict from the surrogate or from the server. For this reason, in the remainder of the paper we analyze only the location of the engine in the server but not in the surrogate.

C. Prefetching Engine

The prefetching engine is aimed at preprocessing those object requests predicted by the prediction engine. By processing the requests in advance, the user's waiting time when the object is actually demanded is reduced. In the literature, this preprocessing has been mainly concentrated on the transference of requested objects in advance [5], [1], [9] although other approaches consider the preprocessing of a request by the server [8], or focus on the pre-establishment of connections to the server [10].

In addition, to avoid the interference of prefetching with the user's requests, the prefetching engine can take into account external factors to decide whether to prefetch an object hinted by the prediction engine or when to prefetch it. These external factors could be related to any part of the web architecture. For instance, in the client some commercial products wait for the user to be idle to start the prefetching [11], [12]. In the proxy, the prefetching engine could prefetch objects only when the bandwidth availability is higher than a given threshold, and the server could prefetch only when its current load permit an increase of requests.

The prefetching engine can be located at the client, at the proxy, at the server, or at several elements. The capability of reducing the user's perceived latency by means of web prefetching depends on where this engine is located. Since the goal of this paper is to analyze the limits of the latency reduction, we assume that the prefetching engine is located at the client because it permits to reduce the whole user's perceived latency. Besides, it is the current trend as it is included in commercial products like Mozilla Firefox [11] and Google Web Accelerator [12].

III. Workload Description

The workload used to quantify the limits of the reduction of web latencies when using prefetching was obtained from the log of the Squid proxy of the Polytechnic University of Valencia. The log includes 43,418,555 web accesses of 9,359 different users to 160,296 different servers. It is 7 days long, from May 31 to Jun 6 2006. In the analysis, the first 6 days were taken as a training set while the trace of the last day was used to calculate performance metrics.

The log consists of one line per each client request with the standard information of the Squid logs. The main fields are the URL, the timestamp of the moment when the request was finished, the HTTP method, the proxy service time, the MIME type, the client IP and the amount of bytes transferred. In order to reproduce the original sequence of accesses made by the users, the trace was reordered by the starting time of the requests. It was calculated by subtracting in each line the proxy service time from the timestamp. For those measurements that require having the log grouped by page accesses, an alternative version of the log was created. To do so, all the embeddable objects (those with mime type like GIF, CSS or JS) were marked as embedded objects of the last HTML accessed by that user. Accesses not related to user's accesses, like automatic antivirus and O. S. updates were removed to avoid interferences with the user's accesses.

IV. Location of the Prediction Engine

There exist two situations in which a user access cannot be predicted. In this section we analyze these situations, which may limit strongly the performance, and define performance indexes to quantify the limits on performance.

The first one is when accessing for the first time to the element of the web architecture in which the prediction engine is located. For instance, a predictor located at the web server cannot predict the first access of a user in a session. We quantify this limitation by means of the *session first-access ratio* (SFAR), described below.

The second situation in which the prediction is limited is when the object has never been seen before by the prediction engine. This is particularly important when the prediction engine is located at the client. We use the *first seen ratio* (FSR) to evaluate the impact of this limiting factor on the predictive performance. This index has two variants, FSR_{object} and $\text{FSR}_{latency}$, depending on whether the limits on performance are measured in number of objects that can be potentially predicted or in potential latency savings. The FSR_{object} is defined as the ratio of the amount of different objects to the total amount of requests (see Eq. 1). When aggregating data from different clients, the FSR_{object} is calculated as the ratio of different pairs {URI, client IP} to the total amount of requests. $\text{FSR}_{latency}$ is defined as the ratio of the sum of the latency of the first appearance of each object to the latency of the total amount of accesses.

$$FSR_{object} = \frac{\#DifferentObjects}{\#ObjectRequests} \quad (1)$$

Like the FSR, the SFAR has different variants depending on how the limiting factor is quantified: object, latency and page. The index uses the *session* concept, defined as a sequence of accesses made by a single user with idle times between them no longer than *t* seconds. Since each session has always a first page, which cannot be predicted, we define the SFAR_{page} as the ratio of the number of sessions to the total amount of page accesses (see Eq. 2). SFAR_{object} is used to quantify the amount of objects that this factor makes it impossible to predict. It is defined as the ratio of session first-accesses to the total amount of object requests, where a session first-access is a client request that is included in the first page of a session. Finally, to measure the impact of the SFAR factor on the perceived latency, the $\text{SFAR}_{latency}$ is calculated as the

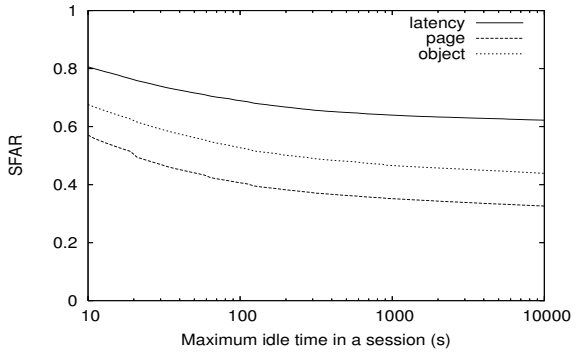

Fig. 1. Server SFAR as a function of the maximum idle time in a session

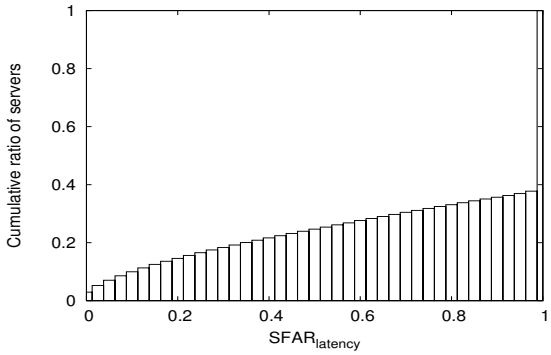

Fig. 2. Cumulative relative frequency histogram of SFAR$_{latency}$ at the server with *t*=1,800 seconds to web servers

ratio of the latency of the session first-accesses to the latency of the total amount of accesses.

$$SFAR_{page} = \frac{\#Sessions}{\#PageAccesses} \quad (2)$$

This section analyzes the impact on the potential performance considering different locations of the prediction engine, since this location makes the pattern of accesses seen by the predictor vary. To this end, the described indexes are evaluated in each prefetching architecture to quantify the two situations in which the predictive performance is limited.

A. Predicting at the server

The main advantage of locating the prediction engine at the server is that it is the element that has most knowledge about the objects that it contains. In addition, it is possible to avoid interferences with the user's requests that prefetching might produce by varying the prediction aggressiveness as a function of the server load. However, from the server point of view, it is not possible to gather dependencies between accesses to other servers, increasing the SFAR measured at the servers and, therefore, decreasing the potential reduction of the user's perceived latency.

Fig. 1 shows the three variants of SFAR metric as a function of the maximum idle time allowed in a session (*t* parameter). As observed, for a *t* value of 1,800 seconds (as used in [13]), the first pages of a session seen from the server point of view (SFAR$_{page}$) represent about 35% of the total pages visited by users. However, these pages contain about 45% of the accessed objects, as shown by the SFAR$_{object}$ curve, but represent 63.4% of the latency perceived by the user, shown by the SFAR$_{latency}$ curve. Therefore, the latency that cannot be reduced due to this factor is 36.6%.

A deep analysis of this result shows that this poor potential reduction of latency is due to the fact that users access most of the servers only once. Fig. 2 shows how the SFAR$_{latency}$ is distributed through the servers. Each point of the curve represents the ratio of servers that have a SFAR$_{latency}$ lower than its corresponding value in the X axis. As one can observe, more than 60% of servers have all their latency in first pages. For this reason, the potential latency reduction is highly dependent on the server workload since, as shown in Fig. 2, there is a noticeable amount of servers that could benefit from prefetching in a higher extent. For instance, 20% of servers have a SFAR$_{latency}$ lower than 37.4% (i.e., a potential latency reduction higher than 62.6%).

TABLE I

FIRST SEEN RATIO AT DIFFERENT SERVERS

Metric	Server				
	A	B	C	D	E
FSR$_{object}$	0.5%	0.8%	0.6%	0.4%	0.6%
FSR$_{latency}$	3.9%	9.3%	1.7%	1.8%	1.7%

It is not possible to infer a precise FSR value at the servers from a proxy trace because it gathers only a part, in most cases not significant, of the requests received by a server. To obtain an approximate value, we calculated the FSR in the top-5 servers with more load gathered in the proxy. Table I shows that, among these servers, the FSR$_{object}$ is lower than 0.8%, although the latency of these objects represent 9.3% of the user's perceived latency.

In summary, the main limitation to reduce the latency when the prediction engine is located at the server is the inability of predicting the first access to the server. This fact makes it impossible to reduce the user's perceived latency beyond 36.6% with this prefetching architecture.

B. Predicting at the client

Locating the predictor engine at the client has two main advantages; i) the client is the element where the SFAR is the lowest since the predictor engine could gather all the user's accesses, and ii) it is possible to gather, and therefore, to predict, cross-server transitions. In contrast, this location has, as main shortcoming, the low amount of accesses that it receives, so making FSR high.

Regarding the prediction limit that affects the prediction of objects never seen before, we found a FSR$_{object}$value of 41.6%, which is the percentage of objects demanded by a user that have never been accessed by that user before. The latency of these objects represent 45.6% of the total latency perceived by users, which is the latency that cannot be reduced when the predictor is located at the client.

Fig. 3 shows the prediction limit related to the session first-access factor. As one can observe, the values are noticeably lower than those measured at the server. For a *t* value of 1,800 seconds, only 1.3% of pages are the first ones in a session, which represent 2.8% of the user's perceived latency.

In summary, unlike the predictor located at the server, the main limitation of the web prefetching techniques for reducing the user's perceived latency is the access to objects never seen before. This fact makes that the maximum latency that this

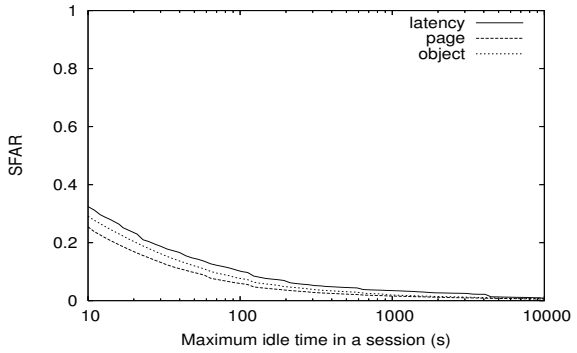

Fig. 3. Client SFAR as a function of the maximum idle time in a session

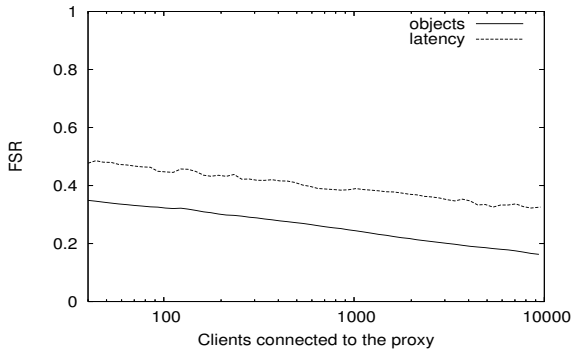

Fig. 4. FSR at the proxy depending on the amount of clients connected to the proxy

web prefetching architecture could reduce is about 54.4% of the latency perceived by users.

C. Predicting at the proxy

Locating the predictor at the proxy has similar advantages than locating it at the client, since both gather all the requests performed by the client. For this reason, their SFAR matches. However, the FSR has a better value in the proxy than in the client, due to the high amount of clients usually connected to the proxy. To analyze how the FSR changes depending on the amount of connected clients, experiments were run varying the number of users connected from 40 to 9,359, randomly selected from the original trace.

Fig. 4 shows that in a small sized proxy with 40 clients connected, 34% of user's accesses were made to an object never requested before. This FSR decreases logarithmically to 16% when considering all the user's connected. It occurs in a similar way when looking at the latency curve, which decreases from 47% with a 40-client proxy to 33% with the 10,000-user proxy.

The values obtained for SFAR and FSR let us state that the first seen objects are the main limiting factor of predictors in the proxy in order to reduce the user's perceived latency.

D. Collaborative prediction

Previous subsections have shown that, depending on the location of the prediction engine, not only the potential latency reduction varies but also the source of this limitation. Therefore, if prediction engines were implemented in several elements of the architecture, they could collaborate to improve the predictions and overcome the limitations.

In this subsection we quantify the latency reduction limits for the different combinations of collaborative prediction engines. Notice that when dealing with collaborative predictors, a client or a proxy can collaborate only with a server if it has been accessed before. To quantify this limitation, we define the *server first seen ratio* (SFSR) as the ratio of accesses to pages of servers that have been accessed for the first time to the total amount of accesses to pages.

1) Collaborative prediction between clients and servers: A collaborative prediction engine that has predictors in the clients and in the servers takes benefit from the low SFAR of the client-located predictors and the good knowledge of the objects of server-located predictors. By analyzing the SFSR, we found that 5.0% of the pages accessed by the user are requested to a server that is accesed for the first time by that user. These requests represent 4.6% of the total user's perceived latency.

As discussed in Section IV-B, the latency that cannot be reduced due to the session first-accesses at clients is 2.8% of the perceived latency, whereas the FSR factor at the server limits the potential latency reduction between 90.7% and 98.3%. Since we quantify the theoretical maximum of the latency reduction, we consider the best situation (i.e., 98.3%).

All three limits provide a latency of 4.6% as a maximum value that cannot be reduced. Therefore, we conclude that a predictor engine based on collaborative predictors located at the clients and at the servers cannot reduce the user's perceived latency more than 95.4%. Remark that this is only a theoretical limit of the capability of the prediction engine to potentially predict objects.

2) Collaborative prediction between proxy and servers: Like the previous scheme of collaboration, this one also takes benefit from the low SFAR of proxy prediction engines and the low FSR of predictors located at servers. The limits related to the session first-access factor and to the first seen objects are the same as the limits analyzed in the collaborative prediction between clients and servers, because the SFAR at the proxy matches the SFAR at the client (see Section IV-C).

The analysis of the trace provided a SFSR_{object} value of 0.8%, that is, only 0.8% of the pages requested by the proxy were made to a server that was not accessed before. These pages represented 0.7% of the latency recorded at the proxy. This result makes the session first-access factor the most restrictive limit to potentially reduce the user's perceived latency. Thus, a collaborative predictor engine located at the servers and at the proxy cannot reduce the user's perceived latency more than 97.2%.

3) Collaborative prediction between clients and proxy: A prediction engine located at the client could take benefit from collaborating with a predictor located at the proxy because the latter has a lower FSR. However, the opposite situation does not occur because the proxy predictor cannot benefit from the collaboration since SFAR matches at both elements Therefore this collaborative scheme cannot outperform a single predictor located at the proxy.

For the same reason, a collaborative system between predictors in all the elements of the web architecture, i.e., clients, proxies and servers, cannot outperform the collaborative scheme between proxies and servers.

TABLE II

SUMMARY OF PREDICTING LIMITS DEPENDING ON WHERE IS LOCATED THE PREDICTION ENGINE

	Single Predictor			**Collaborative Predictors**	
Factor	Client	Proxy	Server	Client-Server	Proxy-Server
$SFAR_{object}$	1.3%	1.3%	45%	1.3%	1.3%
FSR_{object}	41.6%	16.3%	0.4%	0.4%	0.4%
$SFSR_{object}$	–	–	–	5.0%	0.8%
Maximum prediction of objects	58.4%	83.7%	55%	95%	98.7%
$SFAR_{latency}$	2.8%	2.8%	63.4%	2.8%	2.8%
$FSR_{latency}$	45.6%	32.6%	1.7%	1.7%	1.7%
$SFSR_{latency}$	–	–	–	4.6%	0.7%
Maximum reduction of latency	54.4%	67.4%	36.6%	95.4%	97.2%

E. Discussion

Table II summarizes the potential prediction capabilities and the sources of the limits of each prefetching architecture analyzed. As one can observe, collaborative prediction engines could reduce almost all the user's perceived latency. However, latency reduction of single predictors is more limited.

The best element to implement a single predictor is the proxy, which could reduce up to 67% of the perceived latency. The prediction engines located at servers can only reduce 36.6% the overall latency. This is due to the high amount of servers that are accessed only once per session. However, there is a noticeable amount of servers that can benefit from implementing the prediction engine in a higher extent.

Regarding the sources of the latency reduction limits, the session first-accesses are the main limiting factor in server predictors. The first seen objects represent the main limiting factor in client and proxy single predictors, as well as in the collaborative prediction engine of proxies with servers. Finally, the factor of accessing to servers never seen before is the main limit to reduce the user's perceived latency in the collaborative predictors of clients with servers.

V. RELATED WORK

Kroeger *et. al* in [6] analyze the latency reduction limits of caching and prefetching under several scenarios with a prefetching engine located at the proxy by using a proxy trace which dates from 1996. Most of their results are not comparable to the ones presented in this paper because of the different assumptions. The only result that can be compared to those presented in this work is that, for a prediction engine located at the proxy, prefetching could reach a latency reduction of 45% with assumptions similar to those we considered. The comparison of this value to the one presented in this paper; i.e., 67%, illustrates the impact of the changes of the Web during the last years on the potential benefits of prefetching.

Fan *et. al* in [1] also use a proxy-equivalent trace which dates from 1996 to evaluate their proposal of web prefetching between proxies and clients. In their proposal, the proxy implements the predictor and then pushes the predicted objects to the clients when they are idle. The potential performance for their architecture is evaluated by means of a *perfect* prefetching algorithm. Their results show that, by combining prefetching and other techniques, 50% of the requests can be predicted to avoid about 30% of the perceived latency.

VI. CONCLUSIONS

In this paper we described a generic web prefetching architecture and discussed the benefits of the possible locations of the prediction engine on which web prefetching lies. The analysis of the location was focused on the limits of the prediction of object requests as well as on the limits of reducing the user's perceived latency.

The results showed that the best element to locate the prediction engine is potentially the proxy, whose implementation could reduce the perceived latency up to 67.4%. Implementing predictors in clients or servers can also be beneficial, although in a lower or much lower extent, respectively.

Schemes for collaborative predictors located at diverse elements of the web architecture were also analyzed. These schemes largely improve the single predictor at the proxy, since a proxy-server collaborative prediction engine reaches a theoretical limit of reducing 97% of the perceived latency.

REFERENCES

[1] L. Fan, P. Cao, W. Lin, and Q. Jacobson, "Web prefetching between low-bandwidth clients and proxies: Potential and performance." in *Proc. of ACM SIGMETRICS Conf. on Measurement and Modeling Of Computer Systems*, Atlanta, USA, 1999.

[2] C. Bouras, A. Konidaris, and D. Kostoulas, "Predictive prefetching on the web and its potential impact in the wide area." *World Wide Web*, vol. 7, no. 2, pp. 143–179, 2004.

[3] E. Markatos and C. Chronaki, "A top-10 approach to prefetching on the web," in *Proc. of INET' 98*, Geneva, Switzerland, 1998.

[4] J. Domènech, J. A. Gil, J. Sahuquillo, and A. Pont, "DDG: An efficient prefetching algorithm for current web generation," in *Proc. of 1st IEEE Workshop on Hot Topics in Web Systems and Tech.*, Boston, USA, 2006.

[5] R. Kokku, P. Yalagandula, A. Venkataramani, and M. Dahlin, "NPS: A non-interfering deployable web prefetching system," in *Proc. of USENIX Symp. on Internet Tech. and Systems*, Palo Alto, USA, 2003.

[6] T. M. Kroeger, D. D. Long, and J. C. Mogul, "Exploring the bounds of web latency reduction from caching and prefetching," in *Proc. of 1st USENIX Symp. on Internet Tech. and Systems*, Monterey, USA, 1997.

[7] K. Lau and Y.-K. Ng, "A client-based web prefetching management system based on detection theory." in *Proc. of Web Content Caching and Distribution: 9th Intl. Workshop (WCW 2004)*, Beijing, China, 2004.

[8] S. Schechter, M. Krishnan, and M. D. Smith, "Using path profiles to predict http requests," in *Proc. of 7th Intl. World Wide Web Conf.*, Brisbane, Australia, 1998.

[9] B. Wu and A. D. Kshemkalyani, "Objective-optimal algorithms for long-term web prefetching," *IEEE Trans. on Computers*, vol. 55, no. 1, pp. 2–17, 2006.

[10] E. Cohen and H. Kaplan, "Prefetching the means for document transfer: a new approach for reducing web latency," *Computer Networks*, vol. 39, no. 4, pp. 437–455, 2002.

[11] "Link prefetching FAQ," http://mozilla.org/projects/netlib/.

[12] "Google web accelerator," http://webaccelerator.google.com/.

[13] R. Cooley, B. Mobasher, and J. Srivastava, "Data preparation for mining World Wide Web browsing patterns," *Knowledge and information systems*, vol. 1, no. 1, pp. 5–32, 1999.